Mobile Application Development

Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Android Persistence, Multithreading & Networking

Agenda & Goals

- Be aware of the different approaches to data persistence and multithreading in Android Development
- Be able to work with the **`SQLiteOpenHelper`** and **`SQLiteDatabase`** classes to implement an SQLite database on an Android device (to manage our Coffees)
- Be able to work **`AsyncTasks`** and **`Volley`** to execute background tasks and make API calls

CoffeeMate.4.0

Using an SQLite Database

Main Idea – why do we need Persistence?

- Android can shut down and restart your app
  - When you rotate the screen
  - When you change languages
  - When your app is in background and Android is short on memory
  - When you hit the Back button
- Problem
  - You risk losing user changes and data
- Solutions ??

Solutions

- Android provides several options for you to save persistent application data.
- The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.
- Android provides a way for you to expose your private data to other applications — with a `Content Provider`.
  - A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose.

Data Storage Solutions *

- **Shared Preferences**
  - Store private primitive data in key-value pairs.
- Internal Storage
  - Store private data on the device memory.
- External Storage
  - Store public data on the shared external storage.
- **SQLite Databases**
  - Store structured data in a private database.
- Network Connection
  - Store data on the web with your own network server.

Data Storage Solutions *

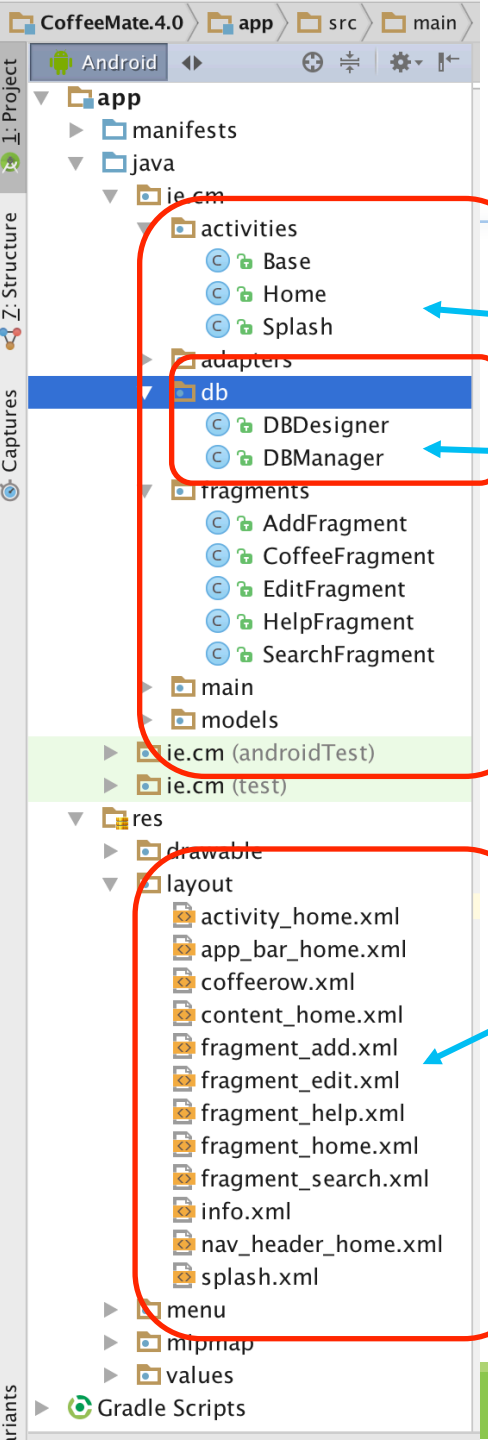
❑ Bundle Class

- A mapping from String values to various `Parcelable` types and functionally equivalent to a standard `Map`.
- Does not handle Back button scenario. App restarts from scratch with no saved data in that case.

❑ File

- Use `java.io.*` to read/write data on the device's internal storage.

CoffeeMate 4.0 – Project Structure



- 15 java source files in total
  - Our Database classes
- xml layouts
- xml menu
- xml files for resources
- xml ‘configuration’ file

Idea

- Goal
  - Enhance **CoffeeMate.3.0** by managing the Coffees in an SQLite Database and improving the UI/UX with a Nav Drawer
- Approach
  - Implement/extend specific classes to add the database functionality to the app – Practical Lab 5

Database Programming in Android *

- Android provides full support for `SQLite` databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.
- The recommended method to create a new SQLite database is to create a subclass of `SQLiteOpenHelper` and override the `onCreate()` method, in which you can execute a SQLite command to create tables in the database.
- For example:

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

Database Programming in Android

- You can then get an instance of your `SQLiteOpenHelper` implementation using the constructor you've defined. To write to and read from the database, call `getWritableDatabase()` and `getReadableDatabase()`, respectively. These both return a `SQLiteDatabase` object that represents the database and provides methods for `SQLite` operations.
- You can execute `SQLite` queries using the `SQLiteDatabase query()` methods, which accept various query parameters, such as the table to query, the projection, selection, columns, grouping, and others. For complex queries, such as those that require column aliases, you should use `SQLiteQueryBuilder`, which provides several convenient methods for building queries.
- Every SQLite query will return a `Cursor` that points to all the rows found by the query. The Cursor is always the mechanism with which you can navigate results from a database query and read rows and columns.

Database Programming in Android

- With `SQLite`, the database is a simple disk file. All of the data structures making up a relational database - tables, views, indexes, etc. - are within this file
- RDBMS is provided through the api classes so it becomes part of your app
- You can use the SQL you learned in a database module
- You should use DB best practices
  - Normalize data
  - Encapsulate database info in helper or wrapper classes
  - Don’t store files (e.g. images or audio), Instead just store the path string

CoffeeMate - DBDesigner

```
public class DBDesigner extends SQLiteOpenHelper
{
    public static final String TABLE_COFFEE = "table_coffee";
    public static final String COLUMN_ID = "coffeeid";
    public static final String COLUMN_NAME = "coffeename";
    public static final String COLUMN_SHOP = "shop";
    public static final String COLUMN_RATING = "rating";
    public static final String COLUMN_PRICE = "price";
    public static final String COLUMN_FAV = "isfavourite";

    private static final String DATABASE_NAME = "coffeemate.db";
    private static final int DATABASE_VERSION = 1;

    // Database creation sql statement
    private static final String DATABASE_CREATE_TABLE_COFFEE = "create table "
            + TABLE_COFFEE + "( " + COLUMN_ID + " integer primary key autoincrement, "
            + COLUMN_NAME + " text not null,"
            + COLUMN_SHOP + " text not null,"
            + COLUMN_PRICE + " double not null,"
            + COLUMN_RATING + " double not null,"
            + COLUMN_FAV + " integer not null);"; //SQLite doesn't support boolean types

    public DBDesigner(Context context) { super(context, DATABASE_NAME, null, DATABASE_VERSION); }

    @Override
    public void onCreate(SQLiteDatabase database) { database.execSQL(DATABASE_CREATE_TABLE_COFFEE); }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(DBDesigner.class.getName(),
                "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_COFFEE);
        onCreate(db);
    }
}
```

Our Table & Column names (for SQL)

Creating the Table (or Tables)

Drop the Table (if we change the schema)

CoffeeMate - DBManager

```
public class DBManager {

    private SQLiteDatabase database;
    private DBDesigner dbHelper;

    public DBManager(Context context) { dbHelper = new DBDesigner(context); }

    public void open() throws SQLException {
        database = dbHelper.getWritableDatabase();
    }

    public void close() { database.close(); }

    public void insert(Coffee c) {
        ContentValues values = new ContentValues();
        values.put(DBDesigner.COLUMN_NAME, c.name);
        values.put(DBDesigner.COLUMN_SHOP, c.shop);
        values.put(DBDesigner.COLUMN_PRICE, c.price);
        values.put(DBDesigner.COLUMN_RATING, c.rating);
        values.put(DBDesigner.COLUMN_FAV, (c.favourite == true) ? 1 : 0);

        long insertId = database.insert(DBDesigner.TABLE_COFFEE, null,
                values);
    }

    public void delete(int id) {...}

    public void update(Coffee c) {...}
```

Our database reference

Returns a reference to the database created from our SQL string

ContentValues are key/value pairs that are used when inserting/ updating databases. Each ContentValue object corresponds to one row in a table

CoffeeMate – DBManager *

```
    public void delete(int id) {...}

    public void update(Coffee c) {...}

    public List<Coffee> getAll() {
        List<Coffee> coffees = new ArrayList<>();
        Cursor cursor = database.rawQuery("SELECT * FROM "
                + DBDesigner.TABLE_COFFEE, null);
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            Coffee pojo = toCoffee(cursor);
            coffees.add(pojo);
            cursor.moveToNext();
        }
        // Make sure to close the cursor
        cursor.close();
        return coffees;
    }

    public Coffee get(int id) {...}

    public List<Coffee> getFavourites() {...}

    private Coffee toCoffee(Cursor cursor) {
        Coffee pojo = new Coffee();
        pojo.coffeeId = cursor.getInt(0);
        pojo.name = cursor.getString(1);
        pojo.shop = cursor.getString(2);
        pojo.price = cursor.getDouble(3);
        pojo.rating = cursor.getDouble(4);
        pojo.favourite = (cursor.getInt(5) == 1) ? true : false;

        return pojo;
    }

    public void setupList() {...}
}
```

This method 'converts' a Cursor object into a Coffee Object

A Cursor provides random read-write access to the resultset returned by a database query

Other Cursor Functions

- moveToPrevious
- getCount
- getColumnIndexOrThrow
- getColumnName
- getColumnNames
- moveToPosition
- getPosition

CoffeeMate.5.0

Multithreading, AsyncTasks & Volley

Background Processes in General

- One of the key features of Android (and iPhone) is the ability to run things in the background
  - Threads
    - Run something in the background while user interacts with UI
  - Services
    - Regularly or continuously perform actions that don't require a UI

Threads

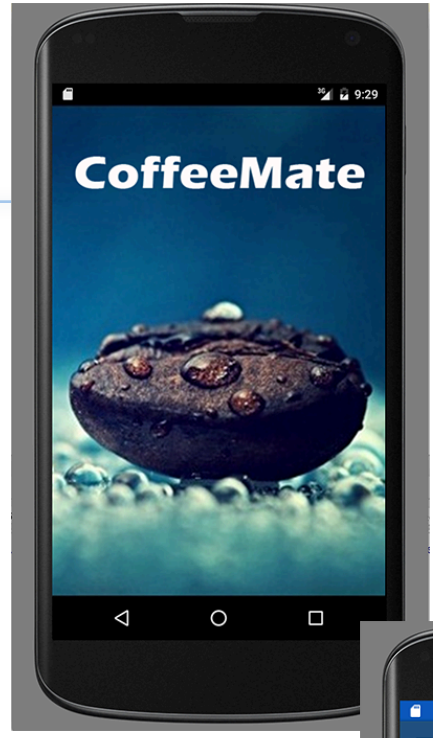
- Recall that Android ensures responsive apps by enforcing a 5 second limit on Activities
- Sometimes we need to do things that take longer than 5 seconds, or that can be done while the user does something else
- Activities, Services, and Broadcast Receivers run on the *main application thread*
- But we can start background/child threads to do other things for us

Android Thread Constraints

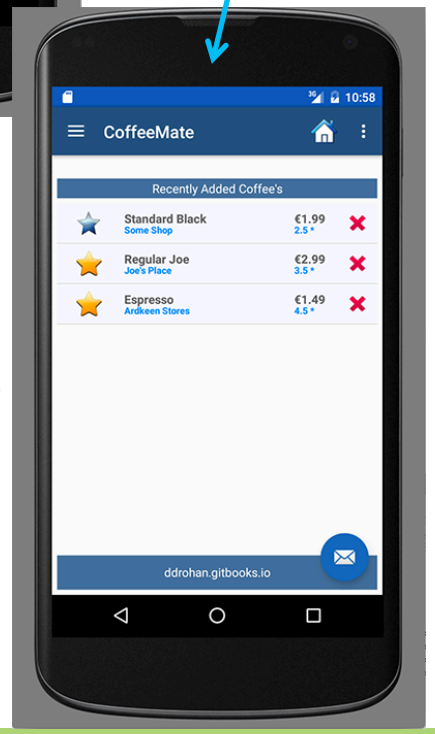
- Child threads ***cannot*** access UI elements (views); these elements must (and can only) be accessed through the main thread
- So what do you do?
  - You pass results to the main thread and let it use the results

Multithreading – Our Splash Screen

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.splash);

    Handler handler = new Handler();
    // run a thread after 2 seconds to start the home screen
    handler.postDelayed(() → {
            // make sure we close the splash screen so the user
            // won't come back when it presses back key
            finish();

            if (!mIsBackButtonPressed) {
                // start the home screen if the back button wasn't pressed already
                Intent intent = new Intent(Splash.this, Home.class);
                Splash.this.startActivity(intent);
            }
    }, SPLASH_DURATION); // time in milliseconds to delay call to run()
}
```



2 secs later



Using the *AsyncTask* class

http://developer.android.com/reference/android/os/AsyncTask.html

- The **AsyncTask** class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

- An asynchronous task is defined by

3 Generic Types	4 Main States	1 Auxiliary Method
Params, Progress, Result	onPreExecute, doInBackground, onProgressUpdate onPostExecute.	publishProgress

Using the *AsyncTask* class

```
AsyncTask <Params, Progress, Result>
```

AsyncTask's generic types
Params: the type of the input parameters sent to the task at execution.
Progress: the type of the progress units published during the background computation.
Result: the type of the result of the background computation.

- Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type **Void**

Note:

Syntax "`String ...`" indicates (Varargs) array of String values, similar to "`String[]`"

Using the *AsyncTask* class

- onPreExecute
  - is invoked ***before*** the execution.
- onPostExecute
  - is invoked ***after*** the execution.
- doInBackground
  - ***the main operation***. Write your heavy operation here.
- onProgressUpdate
  - Indication to the user on the current progress. It is invoked every time publishProgress() is called.

Using the *AsyncTask* class *

```
private class MyAsyncTask extends AsyncTask<String, Void, Bitmap> {
    protected void onPreExecute() {
        // Runs on the UI thread before doInBackground
        // Good for toggling visibility of a progress indicator
        progressBar.setVisibility(ProgressBar.VISIBLE);
    }

    protected Bitmap doInBackground(String... strings) {
        // Some long-running task like downloading an image.
        Bitmap = downloadImageFromUrl(strings[0]);
        return someBitmap;
    }

    protected void onProgressUpdate(Progress... values) {
       // Executes whenever publishProgress is called from doInBackground
       // Used to update the progress indicator
       progressBar.setProgress(values[0]);
    }

    protected void onPostExecute(Bitmap result) {
        // This method is executed in the UIThread
        // with access to the result of the long running task
        imageView.setImageBitmap(result);
        // Hide the progress bar
        progressBar.setVisibility(ProgressBar.INVISIBLE);
    }
}
```

1

2

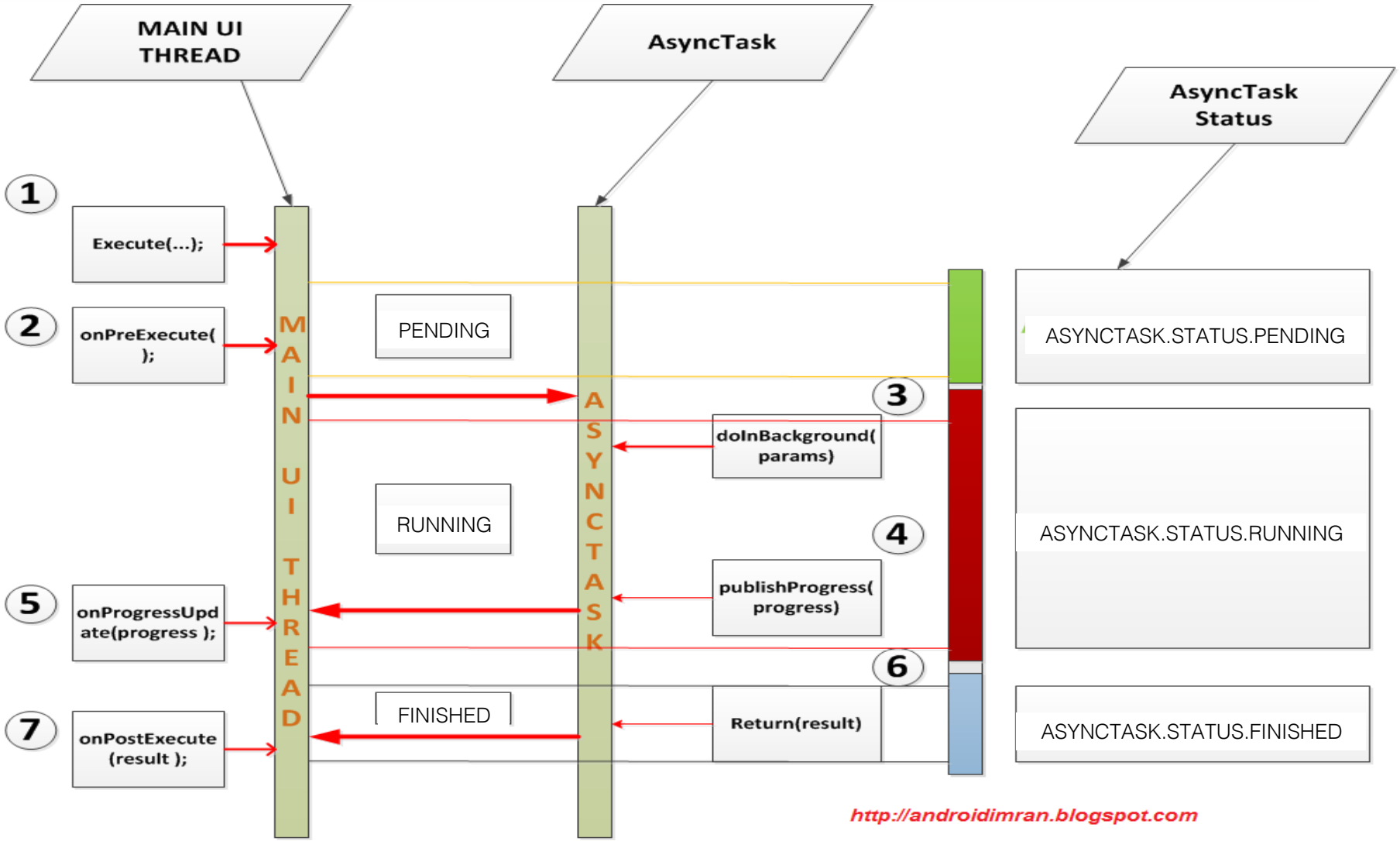
3

4

Using the *AsyncTask* class

AsyncTask's methods
onPreExecute(), invoked on the UI thread immediately after the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.
onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

AsyncTask Lifecycle



CoffeeMate
&
Googles Gson

Google’s Gson

https://sites.google.com/site/gson/gson-user-guide

Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson is an open-source project hosted at http://code.google.com/p/google-gson.

Gson can work with arbitrary Java objects including pre-existing objects that you do not have source-code of.

CoffeeMate & Google's Gson

- To create a POJO from a JSON String we can do something like this (`.fromJson()`)

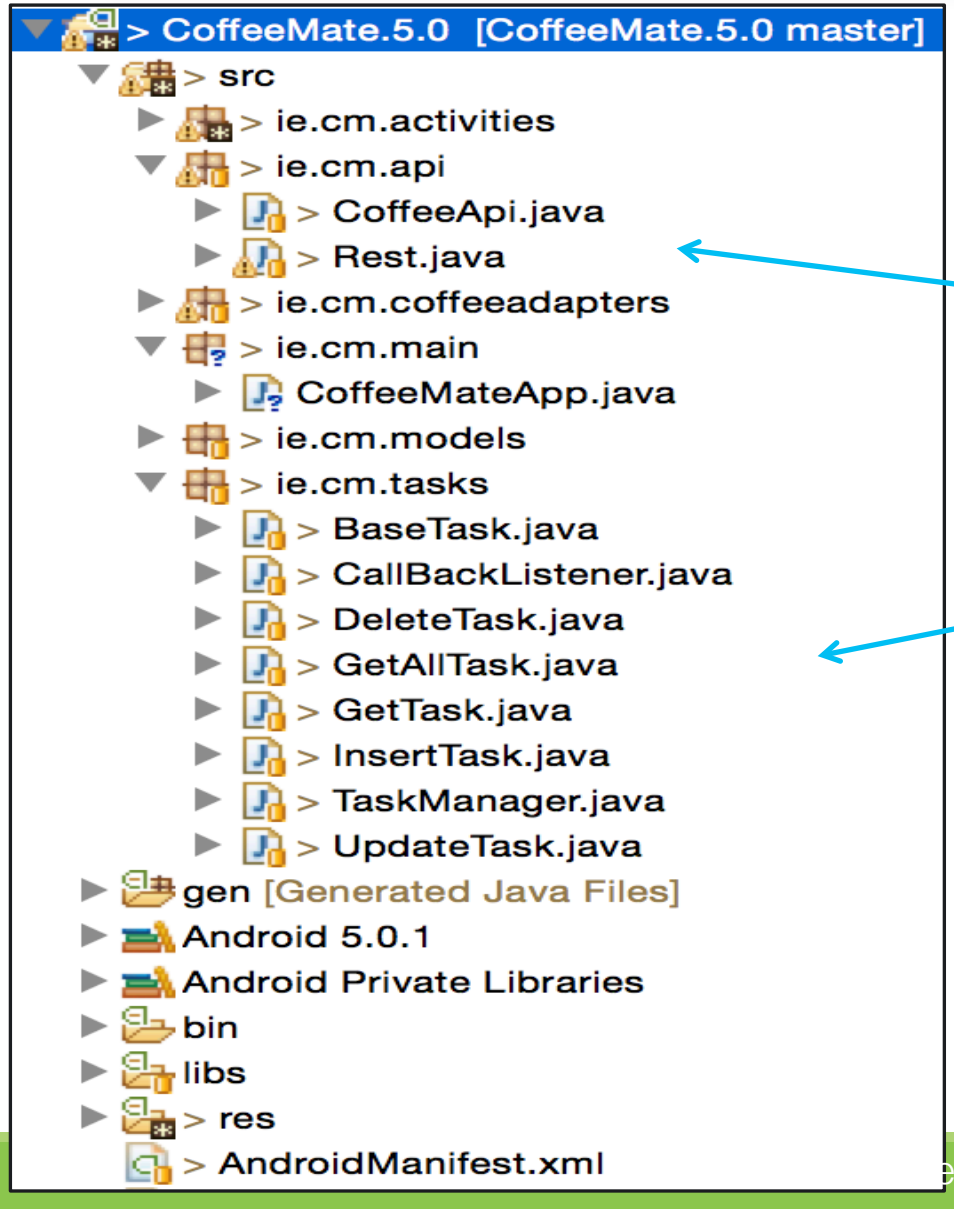
```
// Result handling
Coffee result = null;
Type objType = new TypeToken<Coffee>(){}.getType();
result = new Gson().fromJson(response, objType);
```

- To convert a POJO to a JSON String we can do something like this (`.toJson()`)

```
Type objType = new TypeToken<Coffee>(){}.getType();
String json = new Gson().toJson(aCoffee, objType);
```

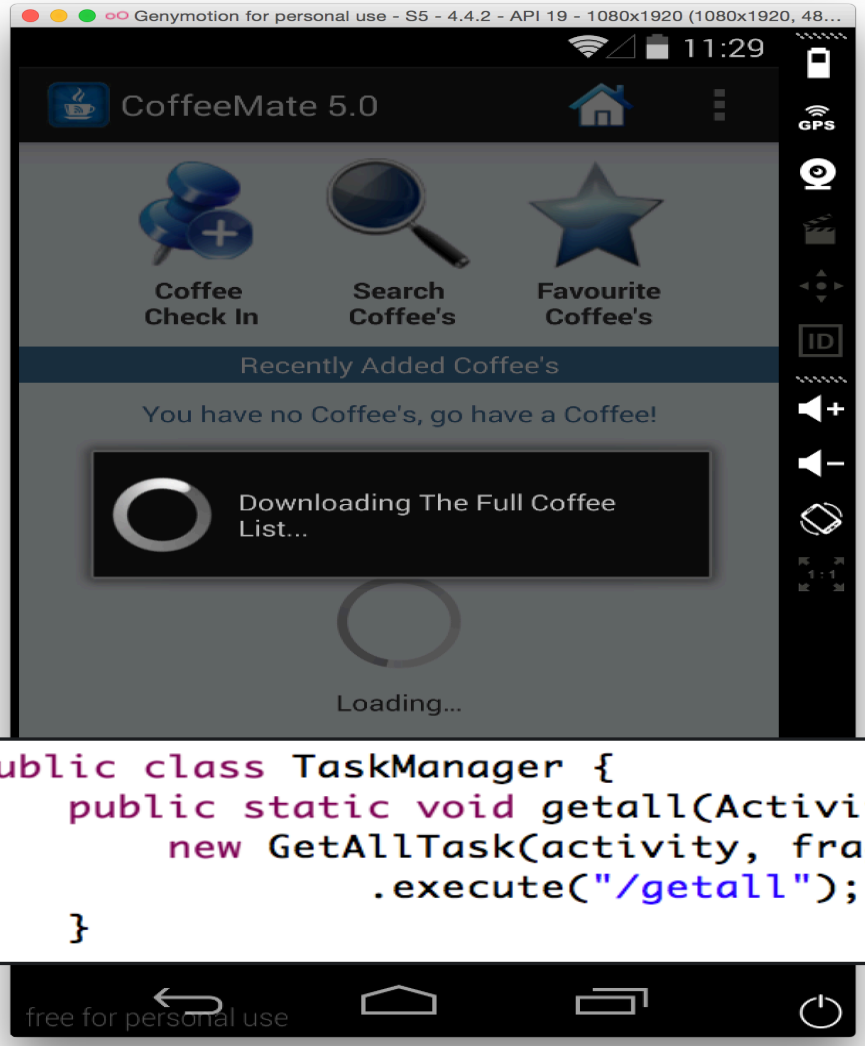
CoffeeMate Example (Using AsyncTasks)

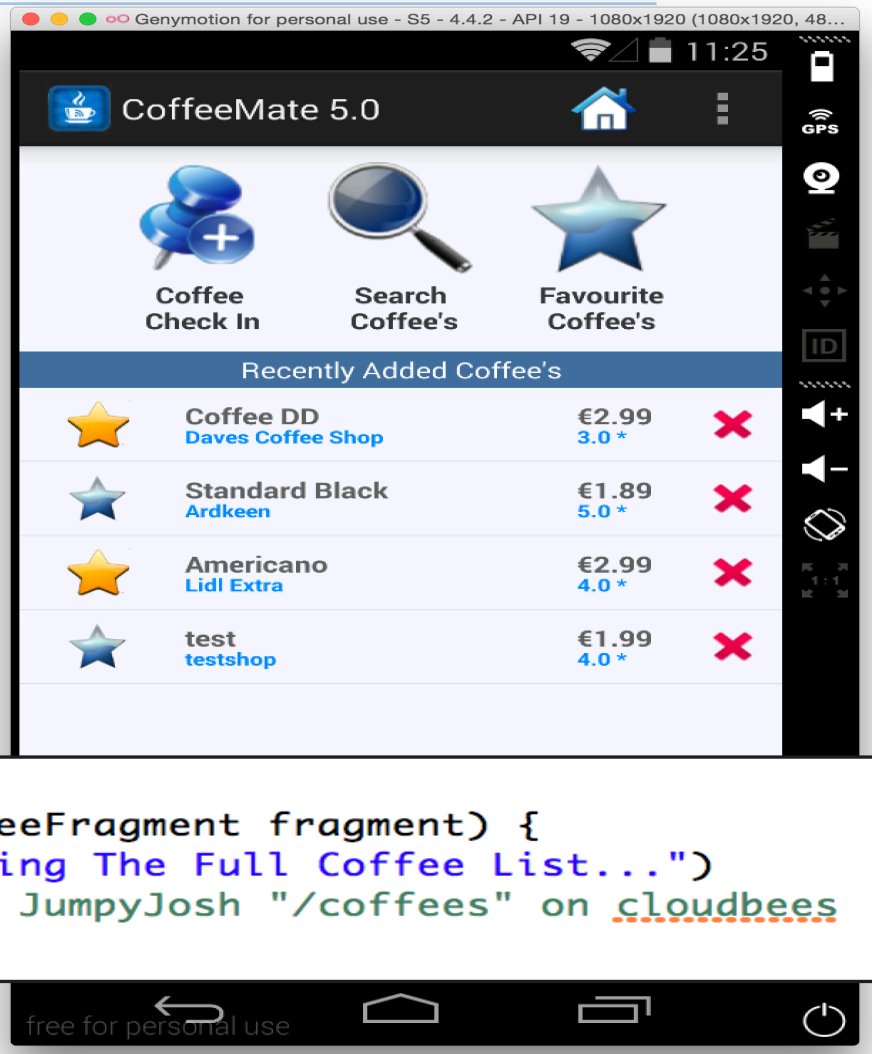
CoffeeMate 5.0 AsyncTasks (and more)



- api classes for calling REST service
- AsyncTasks for CRUD
  (and callback mechanism to update UI)

CoffeeMate 5.0 AsyncTasks






```
public class TaskManager {
    public static void getall(Activity activity,CoffeeFragment fragment) {
        new GetAllTask(activity, fragment, "Downloading The Full Coffee List...")
                .execute("/getall"); // "/getall" on JumpyJosh "/coffees" on cloudbees
    }
```

CoffeeApi

Used for passing/ retrieving JSON coffee data

```
public class CoffeeApi {
    ////////////////////////////////////////////////////////////////////////////
    public static List<Coffee> getAll(String call) {
        String json = Rest.get(call);
        Type collectionType = new TypeToken<List<Coffee>>() {}.getType();

        return new Gson().fromJson(json, collectionType);
    }
    ////////////////////////////////////////////////////////////////////////////
    public static Coffee get(String call) {
        String json = Rest.get(call);
        Type objType = new TypeToken<Coffee>(){}.getType();

        return new Gson().fromJson(json, objType);
    }
    ////////////////////////////////////////////////////////////////////////////
    public static String delete(String call) {
        return Rest.delete(call);
    }
    ////////////////////////////////////////////////////////////////////////////
    public static String insert(String call,Coffee coffee) {
        Type objType = new TypeToken<Coffee>(){}.getType();
        String jsonCoffee = new Gson().toJson(coffee, objType);

        return Rest.post(call,jsonCoffee);
    }
    ////////////////////////////////////////////////////////////////////////////
    public static String update(String call,Coffee coffee) {
        Type objType = new TypeToken<Coffee>(){}.getType();
        String jsonCoffee = new Gson().toJson(coffee, objType);

        return Rest.put(call,jsonCoffee);
    }
    ////////////////////////////////////////////////////////////////////////////
}
```

Rest.java (extract)

```
public class Rest {

    private static String                server;
    private static HttpParams            httpParameters;
    private static DefaultHttpClient     httpClient;

    private static final String URL = "http://www.jumpyjosh.com/cms/rest/api";
    private static final String EURL = "http://coffeemate.edel020.cloudbees.net/api";

    public static void setup() {
        Rest.server = URL;
        httpParameters = new BasicHttpParams();
        HttpConnectionParams.setConnectionTimeout(httpParameters,
                10000);
        HttpConnectionParams.setSoTimeout(httpParameters, 20000);
        httpClient = new DefaultHttpClient(httpParameters);
    }

    private static String getBase() {
        return server;
    }

    ////////////////////////////////////////////////////////////////////////////
    public static String get(String url) {
        String result = "";
        try {
            HttpGet getRequest = new HttpGet(getBase() + url);
            getRequest.setHeader("accept", "application/json");
            //getRequest.setHeader("accept","text/plain");
            HttpResponse response = httpClient.execute(getRequest);
            result = getResult(response).toString();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return result;
    }
```

Used for making the web service calls

AsyncTask - *TaskManager*

```
public class TaskManager {
    public static void getall(Activity activity,CoffeeFragment fragment) {
        new GetAllTask(activity, fragment, "Downloading The Full Coffee List...")
                .execute("/getall"); // "/getall" on JumpyJosh "/coffees" on cloudbees
    }

    public static void get(Activity activity, int id) {
        new GetTask(activity, "Downloading Individual Coffee Data...")
                .execute("/get/" + id);
    }
    public static void delete(Activity activity,CoffeeFragment fragment,int id) {
        new DeleteTask(activity, fragment, "Deleting A Single Coffee...")
                .execute("/delete/" + id);
    }

    public static void insert(Activity activity,Coffee c) {
        new InsertTask(activity, "Adding a Single Coffee...")
                .execute("/insert",c,"dave");
    }

    public static void update(Activity activity,Coffee c) {
        new UpdateTask(activity,"Updating A Single Coffee...")
                .execute("/update",c);
    }
}
```

Calling our AsyncTasks

In our CoffeeFragment

```
@Override
public void onResume() {
    super.onResume();
    TaskManager.getall(getActivity(),this);
}
```

AsyncTask -*BaseTask*

```
public class BaseTask<T> extends AsyncTask<Object, Void, Object> {

    protected CallBackListener<T>    callback;
    protected ProgressDialog         dialog;
    protected Context                context;
    protected String                 message;

    @SuppressWarnings("unchecked")
    public BaseTask(Context context,CoffeeFragment fragment,String message)
    {
        this.callback = (CallBackListener<T>) fragment;
        this.context = context;
        this.message = message;
    }

    @SuppressWarnings("unchecked")
    public BaseTask(Context context,String message)
    {
        if(context instanceof CallBackListener)
            this.callback = (CallBackListener<T>) context;
        this.context = context;
        this.message = message;
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        this.dialog = new ProgressDialog(context, 1);
        this.dialog.setMessage(message);
        this.dialog.show();
    }

    @Override
    protected Object doInBackground(Object... params) {
        return null;
    }
}
```

CallBack Reference

```
public interface CallBackListener<T> {
    public void setList(List<T> aList);
    public void setObject(T object);
    public void updateUI();
}
```

Callback Interface

- Necessary, due to AsyncTasks in separate classes

```
public interface CallBackListener<T> {
    public void setList(List<T> aList);
    public void setObject(T object);
    public void updateUI();
}
```

- Reference maintained in BaseTask

```
protected CallBackListener<T>   callback;
```

- Set in subclass Task, via TaskManager, e.g.

```
public static void get(Activity activity, int id) {
    new GetTask(activity, "Downloading Individual Coffee Data...")
            .execute("/get/" + id);
}
```

```
public class GetTask extends BaseTask<Coffee> {

    public GetTask(Context context, String message) {
        super(context, message);
    }
```

Callback Interface

- Invoked in relevant methods

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    activityInfo = getIntent().getExtras();
    TaskManager.get(this, activityInfo.getInt("coffeeID"));
    setContentView(R.layout.edit);
}
```

In our Edit Activity

```
@Override
protected void onPostExecute(Object result) {
    super.onPostExecute(result);

    callback.setObject((Coffee) result);
    callback.updateUI();

    if (dialog.isShowing())
        dialog.dismiss();
}
```

In our GetTask

Callback Interface

- Overridden in class that implements the interface

```
@Override
public void setObject(Coffee object) {
    this.aCoffee = object;
}

@Override
public void updateUI() {
    setTextViewString(R.id.coffeeNameTextView, aCoffee.name);
    setTextViewString(R.id.coffeeShopTextView, aCoffee.shop);

    setEditString(R.id.nameEditText, aCoffee.name);
    setEditString(R.id.shopEditText, aCoffee.shop);
    setEditDouble(R.id.priceEditText, aCoffee.price);
    setRatingBarValue(R.id.coffeeRatingBar, (float)aCoffee.rating);

    favouriteImage = (ImageView) findViewById(R.id.favouriteImageView);

    if (aCoffee.favourite == 1) {
        favouriteImage.setImageResource(R.drawable.ic_favourite_on);
        isFavourite = true;
    } else {
        favouriteImage.setImageResource(R.drawable.ic_favourite_off);
        isFavourite = false;
    }
}
```

In our Edit Activity

AsyncTask -*GetAllTask*

```
public class GetAllTask extends BaseTask<Coffee> {

    public GetAllTask(Context context, CoffeeFragment fragment, String message) {
        super(context, fragment, message);
    }

    @Override
    protected List<Coffee> doInBackground(Object... params) {

        try {
            return CoffeeApi.getAll((String) params[0]);
        }

        catch (Exception e) {
            Log.v("ASYNC", "ERROR : " + e);
            e.printStackTrace();
        }
        return null;
    }

    @SuppressWarnings("unchecked")
    @Override
    protected void onPostExecute(Object result) {
        super.onPostExecute(result);

        callback.setList((List<Coffee>) result);
        callback.updateUI();

        if (dialog.isShowing())
            dialog.dismiss();
    }
}
```

Remaining Tasks implemented in a similar fashion

CoffeeFragment (Extracts)

```
public class CoffeeFragment extends ListFragment implements OnClickListener,
                                    CallBackListener<Coffee> {

    protected static CoffeeListAdapter  listAdapter;
    protected CoffeeFilter              coffeeFilter;
    protected List <Coffee>             coffeeList = new ArrayList<Coffee>();
```

```
    @Override
    public void setList(List<Coffee> aList) {
        if(aList != null) // App Restart....
            coffeeList = (ArrayList<Coffee>) aList;
    }

    @Override
    public void setObject(Coffee object) {}

    @Override
    public void updateUI() {
        listAdapter = new CoffeeListAdapter(getActivity(), this, coffeeList);
        coffeeFilter = new CoffeeFilter(coffeeList, "all", listAdapter);

        if (getActivity() instanceof Favourites) {
            coffeeFilter.setFilter("favourites");
            coffeeFilter.filter(null);
            listAdapter.notifyDataSetChanged();
        }
        this.setListAdapter(listAdapter);
    }
```

Overriding the necessary methods from the interface

Android Networking (Using Volley)

Volley is an HTTP library developed by Google that makes networking for Android apps easier and most importantly, faster. Volley is available through the open AOSP repository.

Introduced during Google I/O 2013, it was developed because of the absence, in the Android SDK, of a networking class capable of working without interfering with the user experience.

Volley

- Volley offers the following benefits:
  - Automatic scheduling of network requests.
  - Multiple concurrent network connections.
  - Transparent disk and memory response caching with standard HTTP cache coherence.
  - Support for request prioritization.
  - Cancellation request API. You can cancel a single request, or you can set blocks or scopes of requests to cancel.
  - Ease of customization, for example, for retry and backoff.
  - Strong ordering that makes it easy to correctly populate your UI with data fetched asynchronously from the network.
  - Debugging and tracing tools.

Why Volley?

- Avoid `HttpUrlConnection` and `HttpClient`
  - On lower API levels (mostly on Gingerbread and Froyo), HttpUrlConnection and HttpClient are far from being perfect. There are some known issues and bugs that were never fixed.
  - Moreover, HttpClient was deprecated in the last API update (API 22), which means that it will no longer be maintained and may be removed in a future release.
  - These are sufficient reasons for deciding to switch to a more reliable way of handling your network requests.

Why Volley?

❑ Avoid `AsyncTask`

- Since the introduction of Honeycomb (API 11), it's been mandatory to perform network operations on a separate thread, different from the main thread. This substantial change led the way to massive use of the `AsyncTask<Params, Progress, Result>` specification.
- The class is pretty straightforward, way easier than the implementation of a `service`, and comes with a ton of examples and documentation.
- The main problem (next slide), however, is the serialization of the calls. Using the `AsyncTask` class, you can't decide which request goes first and which one has to wait. Everything happens FIFO, first in, first out.

Problem Solved…

- The problems arise, for example, when you have to load a list of items that have attached a thumbnail. When the user scrolls down and expects new results, you can't tell your activity to first load the JSON of the next page and only then the images of the previous one. This can become a serious user experience problem in applications such as Facebook or Twitter, where the list of new items is more important than the thumbnail associated with it.
- Volley aims to solve this problem by including a powerful cancellation API. You no longer need to check in `onPostExecute` whether the activity was destroyed while performing the call. This helps avoiding an unwanted `NullPointerException`.

Why Volley?

It's Much Faster

- Some time ago, the Google+ team did a series of performance tests on each of the different methods you can use to make network requests on Android. Volley got a score up to ten times better than the other alternatives when used in RESTful applications.

Small Metadata Operations

- Volley is perfect for small calls, such as JSON objects, portions of lists, details of a selected item, and so on. It has been devised for RESTful applications and in this particular case it gives its very best.

Why Volley?

❑ It Caches Everything

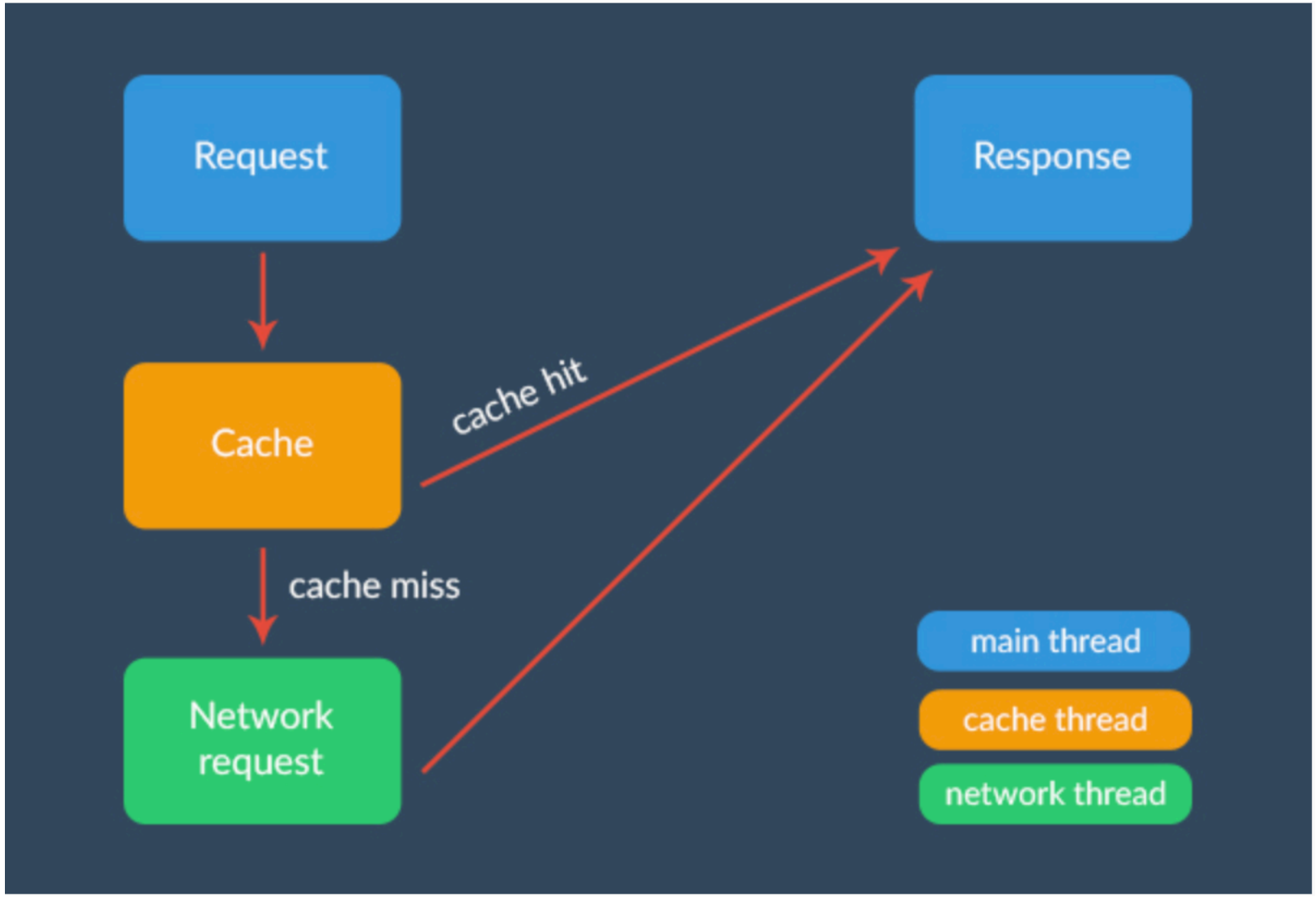
- Volley automatically caches requests and this is something truly life-saving. Let's return for a moment to the example given earlier. You have a list of items—a JSON array let's say—and each item has a description and a thumbnail associated with it. Now think about what happens if the user rotates the screen: the activity is destroyed, the list is downloaded again, and so are the images. Long story short, a significant waste of resources and a poor user experience.
- Volley proves to be extremely useful for overcoming this issue. It *remembers* the previous calls it did and handles the activity destruction and reconstruction. It caches everything without you having to worry about it.

Why Not Volley?

- It is not so good, however, when employed for streaming operations and large downloads. Contrary to common belief, Volley's name doesn't come from the sport dictionary. It’s rather intended as repeated bursts of calls, grouped together. It's somehow intuitive why this library doesn't come in handy when, instead of a volley of arrows, you want to fire a cannon ball.

Under the Hood

- Volley works on three different levels with each level operating on its own thread.



Under the Hood

Main Thread

- On the main thread, consistently with what you already do in the AsyncTask specification, you are only allowed to fire the request and handle its response. Nothing more, nothing less.
- The main consequence is that you can actually ignore everything that was going on in the doInBackground method. Volley automatically manages the HTTP transactions and the catching network errors that you needed to care about before.

Under the Hood

Cache and Network Threads

- When you add a request to the queue, several things happens under the hood. First, Volley checks if the request can be serviced from cache. If it can, the cached response is read, parsed, and delivered. Otherwise it is passed to the network thread.

- On the network thread, a round-robin with a series of threads is constantly working. The first available network thread dequeues the request, makes the HTTP request, parses the response, and writes it to cache. To finish, it dispatches the parsed response back to the main thread where your listeners are waiting to handle the result.

Getting Started With Volley

- Download the Volley Source
  - `git clone https://android.googlesource.com/platform/frameworks/volley`
- Import Source as **`Module`**
  - File -> New Module, choose Import Existing Project
  - Add dependency `compile project(':volley')`

- Alternative – *unofficial* mirror site so beware
  - `compile 'com.mcxiaoke.volley:library-aar:1.0.15'`

Using Volley

- Volley mostly works with just two classes, **`RequestQueue`** and **`Request`**. You first create a **`RequestQueue`**, which manages worker threads and delivers the parsed results back to the main thread. You then pass it one or more **`Request`** objects.

- The **`Request`** constructor always takes as parameters the method type (GET, POST, etc.), the URL of the resource, and event listeners. Then, depending on the type of request, it may ask for some more variables.

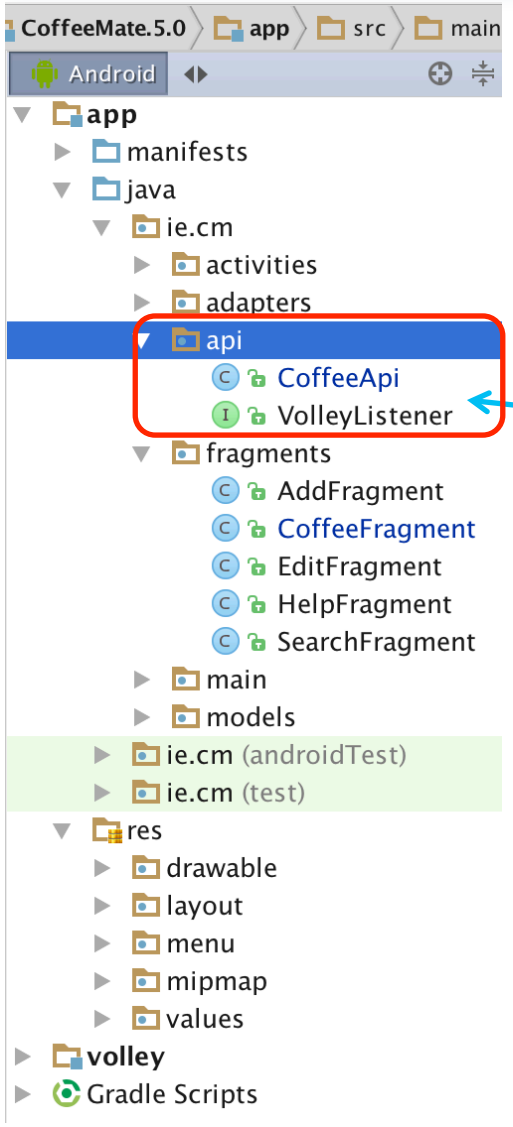
Using Volley *

- Here we create a **`RequestQueue`** object by invoking one of Volley's convenience methods, **`Volley.newRequestQueue`**. This sets up a **`RequestQueue`** object, using default values defined by Volley.
- As you can see, it's incredibly straightforward. You create the request and add it to the request queue. And you're done.
- If you have to fire multiple requests in several activities, you should avoid using this approach - better to instantiate one shared request queue and use it across your project (CoffeeMate 5.0)

```
String url = "http://httpbin.org/html";

// Request a string response
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {

        // Result handling
        System.out.println(response.substring(0,100));

    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {

        // Error handling
        System.out.println("Something went wrong!");
        error.printStackTrace();

    }
});

// Add the request to the queue
Volley.newRequestQueue(this).add(stringRequest);
```

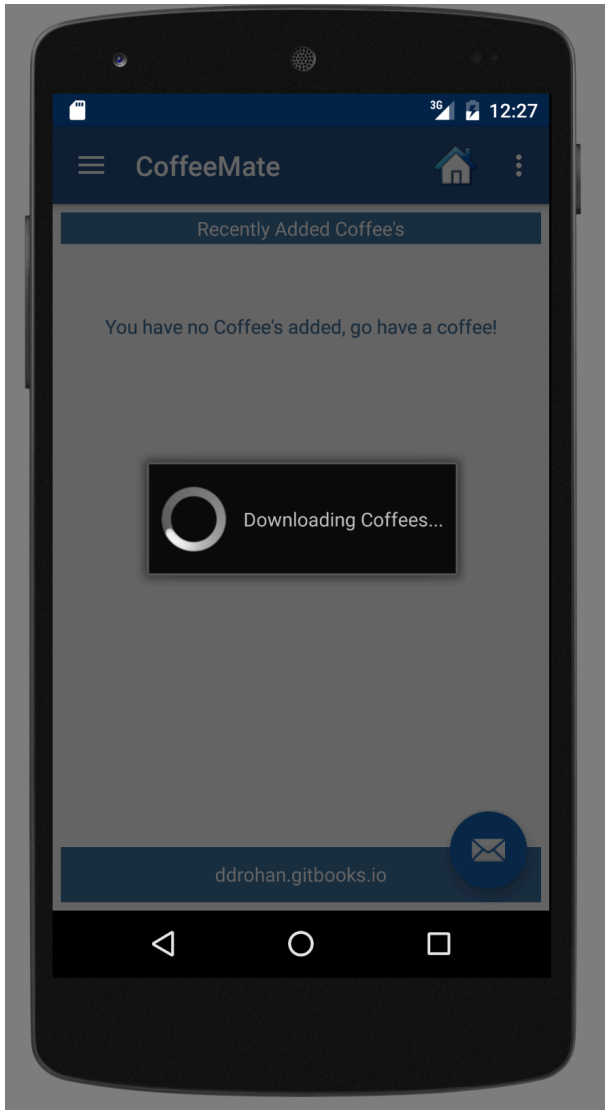
CoffeeMate Example (Using Volley)

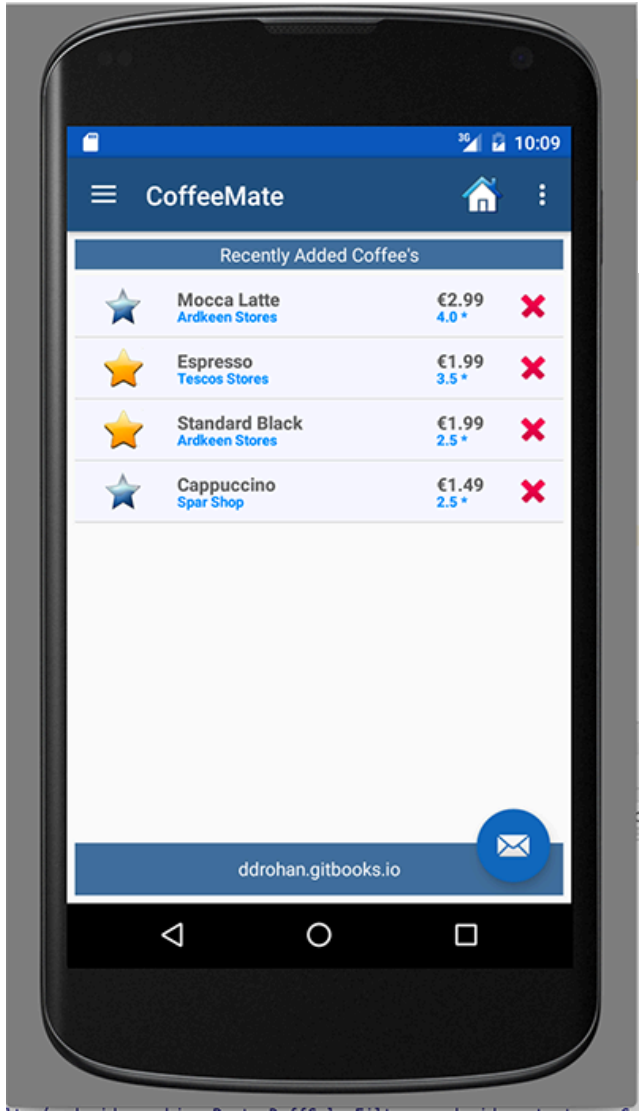
CoffeeMate 5.0 API & Callback Interface



- api class for calling REST service
- callback mechanism to update UI

CoffeeMate 5.0 & Volley






```
@Override
public void onResume() {
  super.onResume();
  //updateUI(this);
  CoffeeApi.attachListener(this);
  CoffeeApi.getAll("/coffees/" + Base.googleToken, mSwipeRefreshLayout);
}
```

- Here we 'attach' our **`VolleyListener`** to the Fragment (**`CoffeeFragment`**) and then *getAll()* of the current users coffees.
- This method triggers a call to *setList()* via the callback interface, which in turn updates the UI ONLY when our API call completes.

```
@Override
public void setList(List list) {
  Base.app.coffeeList = list;
  updateUI(this);
}
```

- We use a similar approach for Updating, Deleting etc.

CoffeeApi – refactored with Volley *

```
public class CoffeeApi {

    private static final String hostURL = "http://coffeemateweb.herokuapp.com";
    private static final String LocalhostURL = "http://192.168.0.13:3000";
    private static       VolleyListener vListener;
    private static final String TAG = "coffeemate";

    public static void attachListener(VolleyListener fragment) { vListener = fragment; }
    public static void detachListener() { vListener  = null; }
    private static void showDialog(String message) {...}
    private static void hideDialog() {...}
    ////////////////////////////////////////////////////////////////////////////////
    public static void getAll(String url, final SwipeRefreshLayout mSwipeRefreshLayout) {
        Log.v(TAG, "GETing from " + url);
            showDialog("Downloading Coffees...");
        // Request a string response
        StringRequest stringRequest = new StringRequest(Request.Method.GET, hostURL + url,
                new Response.Listener<String>() {
                    @Override
                    public void onResponse(String response) {
                        // Result handling
                        List<Coffee> result = null;
                        Type collectionType = new TypeToken<List<Coffee>>(){}.getType();
                        result = new Gson().fromJson(response, collectionType);
                        vListener.setList(result);
                        mSwipeRefreshLayout.setRefreshing(false);
                        hideDialog();
                    }
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                // Error handling
                System.out.println("Something went wrong!");
                mSwipeRefreshLayout.setRefreshing(false);
                error.printStackTrace();
            }
        });
        // Add the request to the queue
        Base.app.add(stringRequest);
    }
```

- Here we create a **`StringRequest GET`** request.
- On a successful **`RESPONSE`** we convert the result into a List of coffees and
- Trigger the callback to set the list in the fragment (and cancel the refresh spinner)

```
@Override
public void setList(List list) {
  Base.app.coffeeList = list;
  updateUI(this);
}
```

CoffeeFragment (Extracts) *

```
public class CoffeeFragment  extends Fragment implements AdapterView.OnItemClickListener,
                                                          View.OnClickListener,
                                                          VolleyListener
```

```
@Override
public void onResume() {
  super.onResume();
  //updateUI(this);
  CoffeeApi.attachListener(this);
  CoffeeApi.getAll("/coffees/" + Base.googleToken, mSwipeRefreshLayout);
}

@Override
public void onPause() {
  super.onPause();
  CoffeeApi.detachListener();
}
```

Overriding the necessary methods from the interface

```
@Override
public void setList(List list) {
  Base.app.coffeeList = list;
}

@Override
public void updateUI(Fragment fragment) {
  fragment.onResume();
}
```

CoffeeMate 5.0 – Using AsyncTasks Vs Volley

- Using AsyncTasks
  - CoffeeApi
  - CallBackListener
  - Rest
  - TaskManager
  - CRUD Tasks x 6
- **Total = 10 Classes**

- Using Volley
  - CoffeeApi
  - VolleyListener
- **Total = 2 Classes**

Summary

- We looked at data persistence and multithreading in Android Development and how to use an SQLite database
- We covered a brief overview of JSON & Googles Gson
- We covered in detail the use of **`AsyncTasks`** and **`Volley`** to execute background tasks and make API calls
- We Compared the two in our CoffeeMate Case Study

References

- Victor Matos Notes – Lesson 13 (Cleveland State University)
- Android Developers
  http://developer.android.com/index.html
- http://code.tutsplus.com/tutorials/an-introduction-to-volley--cms-23800

Questions?

Appendix

- ❑ Multithreading Overview
- ❑ Using a Splash & Login Screen
- ❑ Files
- ❑ Content Providers
- ❑ REST

- ❑ And a bit on Bundles…

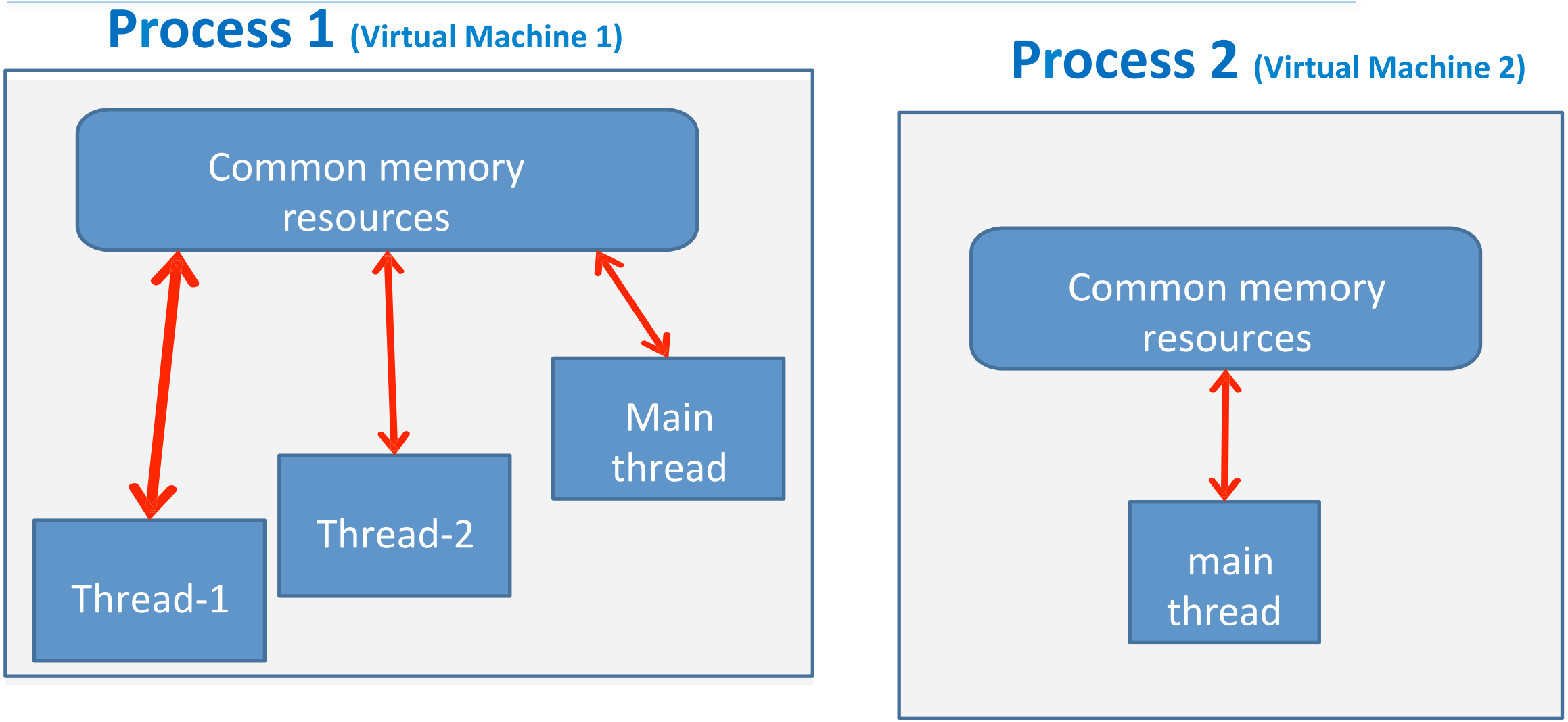
Multithreading Overview

Threads

http://developer.android.com/reference/java/lang/Thread.html

- A Thread is a concurrent unit of execution.
- Each thread has its own call stack. The call stack is used on method calling, parameter passing, and storage for the called method's local variables.
- Each virtual machine instance has at least one main thread.
- Threads in the same VM interact and synchronize by the use of shared objects and monitors associated with these objects.

Threads

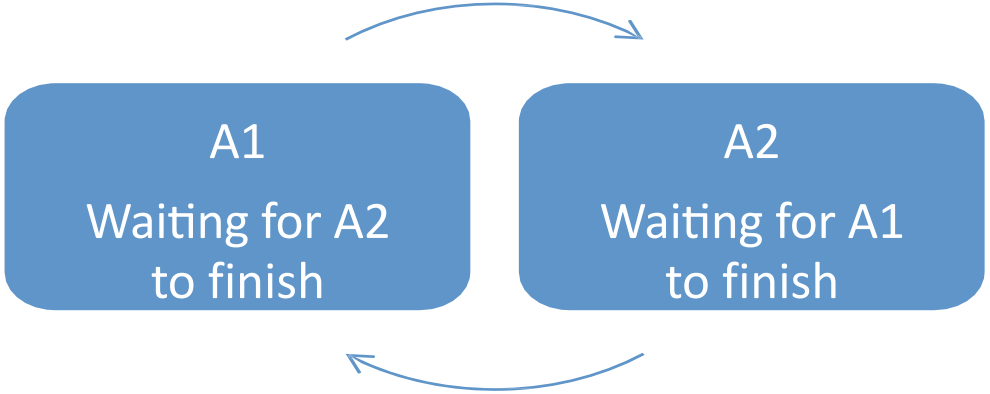

Process 1 (Virtual Machine 1)
Common memory resources
Thread-1
Thread-2
Main thread
Process 2 (Virtual Machine 2)
Common memory resources
main thread

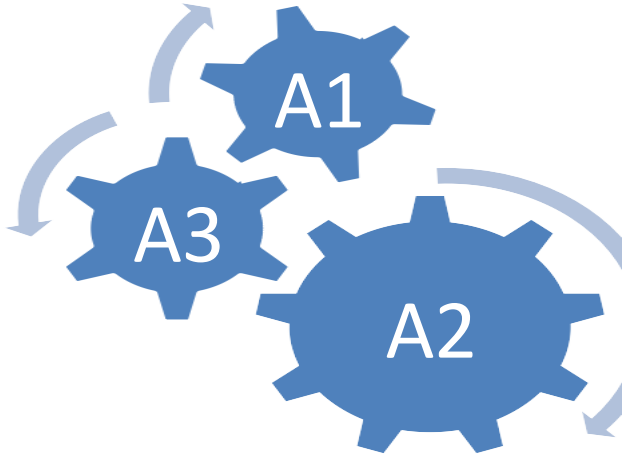
Advantages of Multithreading

- Threads share the process' resources but are able to execute independently.
- Applications responsibilities can be separated
  - main thread runs UI, and
  - slow tasks are sent to background threads.
- Threading provides an useful abstraction of *concurrent* execution.
- A multithreaded program operates *faster* on computer systems that have *multiple CPUs*.
  (Java 8 supports multi-core multi-threading)

Disadvantages

- Code tends to be more complex
- Need to detect, avoid, resolve deadlocks





Android's Approach to Slow Activities

Problem: An application may involve a time-consuming operation.
Goal: We want the **UI** to be responsive to the user in spite of heavy load.
Solution: Android offers two ways for dealing with this scenario:

1. Do expensive operations in a background ***service***, using *notifications* to inform users about next step

2. Do the slow work in a ***background thread***.

Using Threads: Interaction ***between*** Android threads is accomplished using
(a) a main thread ***Handler*** object and
(b) posting ***Runnable*** objects to the main view.

Thread Execution – Example

There are basically two main ways of having a **Thread** execute application code.

- Create a new class that *extends* **Thread** and override its **run()** method.

```
MyThread t = new MyThread();
t.start();
```

- Create a new **Thread** instance passing to it a **Runnable** object.

```
Runnable myRunnable1 = new
MyRunnableClass(); Thread t1 = new
Thread(myRunnable1); t1.start();
```

In both cases, the **start()** method must be called to actually execute the new Thread.

Using a Splash Screen
&
Login Screen

What do we want exactly?

- Display Splash Screen for a few seconds
- Display Login Screen
- Only show Home Screen once valid details entered

Splash

```
public class Splash extends Activity {
    // used to know if the back button was pressed in the splash screen activity
    // and avoid opening the next activity
    private boolean              mIsBackButtonPressed;
    private static final int     SPLASH_DURATION = 2000; // 2 seconds

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash);

        Handler handler = new Handler();
        // run a thread after 2 seconds to start the home screen
        handler.postDelayed(new Runnable() {

            @Override
            public void run() {
                // make sure we close the splash screen so the user
                // won't come back when it presses back key
                finish();

                if (!mIsBackButtonPressed) {
                    // start the home screen if the back button wasn't pressed already
                    Intent intent = new Intent(Splash.this, Login.class);
                    Splash.this.startActivity(intent);
                }
            }
        }, SPLASH_DURATION); // time in milliseconds to delay call to run()
    }

    @Override
    public void onBackPressed() {
        // set the flag to true so the next activity won't start up
        mIsBackButtonPressed = true;
        super.onBackPressed();
    }
}
```

Handler object associated with single thread

Start Login Screen via Intent

Update Manifest File

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ie.cm"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="14" android:targetSdkVersion="21" />
   <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher1"
        android:theme="@style/appTheme"
        android:label="@string/appName"
        android:name="ie.cm.main.CoffeeMateApp">"

        <activity
            android:name="ie.cm.activities.Splash"
            android:configChanges="orientation|keyboardHidden"
            android:screenOrientation="portrait"
            android:theme="@android:style/Theme.NoTitleBar" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    <activity android:name="ie.cm.activities.Home"></activity>
    <activity android:name="ie.cm.activities.Login"></activity>
    <activity android:name="ie.cm.activities.Help"></activity>
    <activity android:name="ie.cm.activities.Add"></activity>
    <activity android:name="ie.cm.activities.Edit"></activity>
    <activity android:name="ie.cm.activities.Search"></activity>
    <activity android:name="ie.cm.activities.Favourites"></activity>
  </application>
</manifest>
```

Activity to Launch

Using SharedPreferences

SharedPreferences (1)

- Two forms:
  - Share across all components in an application
    - getSharedPreferences("SomeString",Activity.MODE_PRIVATE);
  - Store only data needed by this Activity
    - getPreferences(Activity.MODE_PRIVATE);
- Store only data needed by this Activity when Activity becomes inactive (but not when finished)
  - Eg. Orientation change from portrait to landscape
  - use Bundle in onSaveInstanceState / onRestoreInstanceState / onCreate

SharedPreferences (2)

- Create your SharedPreferences instance

```
SharedPreferences settings
= this.getSharedPreferences("Demo",
MODE_PRIVATE);
```

- Add data in the form : <String Key,String Value>

```
SharedPreferences.Editor editor =
editor.putString("name","value");
settings.edit();
editor.commit();
```

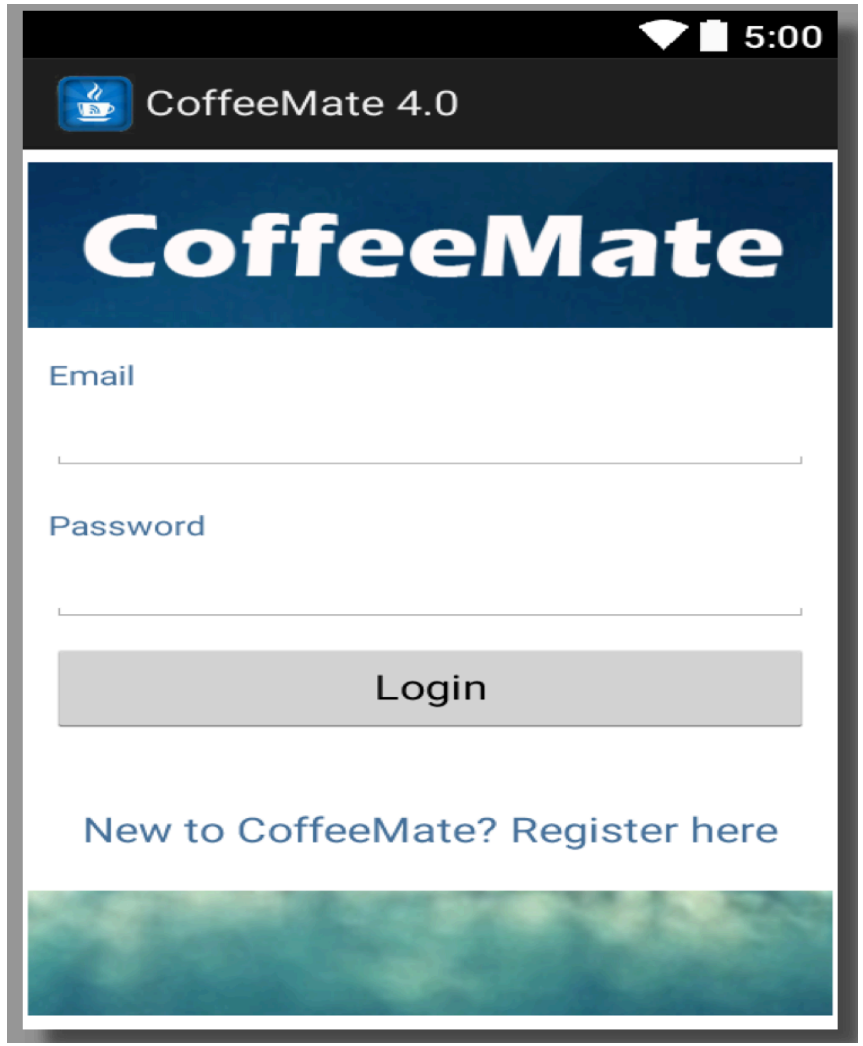
- Use ‘Key’ to get ‘Value’

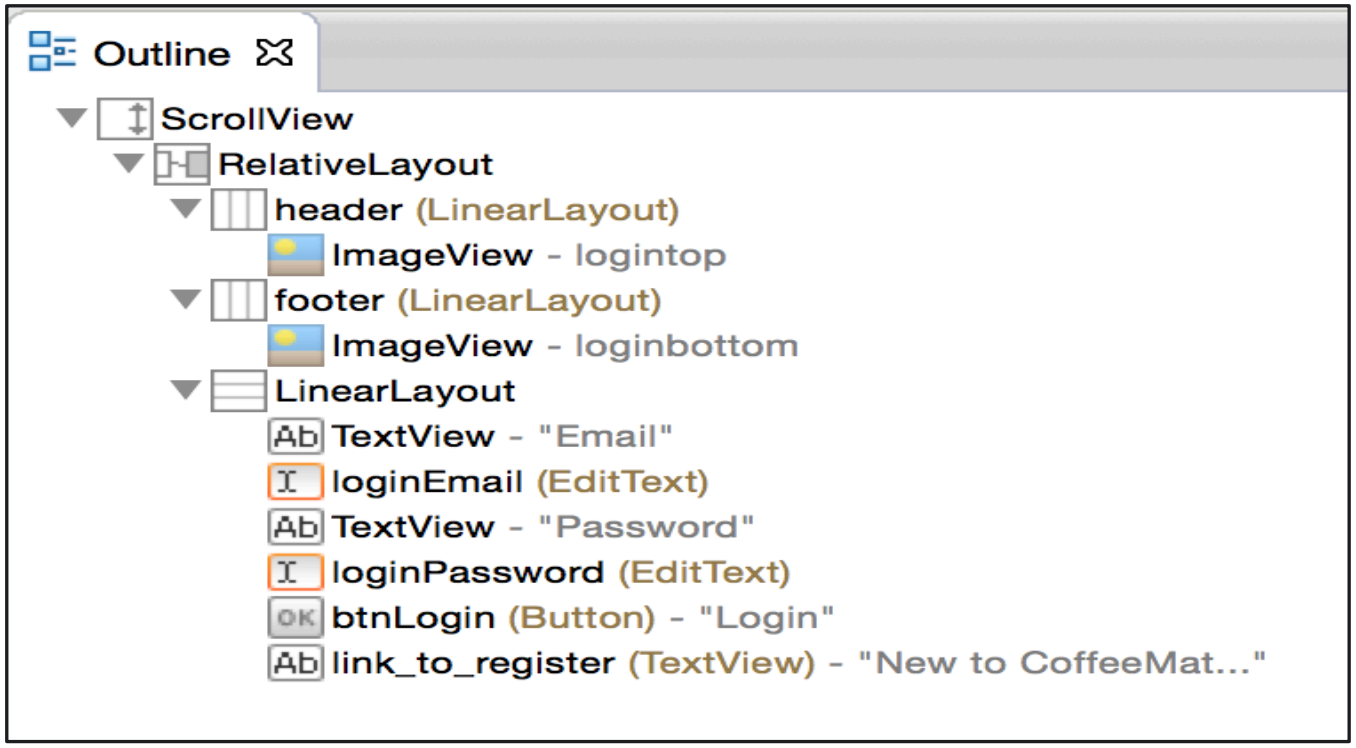
```
String str = settings.getString("name",
"defaultValue");
```

- Reset the preferences (clear)

```
editor.clear().commit();
```

login.xml


5:00
CoffeeMate 4.0
CoffeeMate
Email
Password
Login
New to CoffeeMate? Register here


Outline
ScrollView
RelativeLayout
header (LinearLayout)
ImageView - logintop
footer (LinearLayout)
ImageView - loginbottom
LinearLayout
TextView - "Email"
loginEmail (EditText)
TextView - "Password"
loginPassword (EditText)
btnLogin (Button) - "Login"
link_to_register (TextView) - "New to CoffeeMat..."

Login (1)

```
public class Login extends Activity implements OnClickListener {

    // used to know if the back button was pressed in the splash s
    // and avoid opening the next activity
    private boolean mIsBackButtonPressed;
    private SharedPreferences settings;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        settings = getSharedPreferences("loginPrefs", 0);
        if (settings.getBoolean("loggedin", false))
            /* The user has already logged in, so start the Home S
            startHomeScreen();

        setContentView(R.layout.login);
        Button login = (Button) findViewById(R.id.btnLogin);
        login.setOnClickListener(this);
    }

    @Override
    public void onBackPressed() {
        // set the flag to true so the next activity won't start u
        mIsBackButtonPressed = true;
        super.onBackPressed();
    }
```

Load your Preferences

Default value

Login (2)

```
@Override
public void onClick(View arg0) {

    CharSequence email = ((TextView) findViewById(R.id.loginEmail))
            .getText();
    CharSequence password = ((TextView) findViewById(R.id.loginPassword))
            .getText();

    if (email.length() <= 0 || password.length() <= 0)
        Toast.makeText(this, "You must enter an email & password",
                Toast.LENGTH_SHORT).show();
    else if (!email.toString().matches("d")
            || !password.toString().matches("d"))
        Toast.makeText(this, "Unable to validate your email & password",
                Toast.LENGTH_SHORT).show();
    else if (!mIsBackButtonPressed) {
        // Validate User with Server Here

        // Update logged in preferences
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean("loggedin", true);
        editor.commit();
        // start the home screen if the back button wasn't pressed already
        startHomeScreen();
        this.finish(); // destroy the Login Activity
    }
}

private void startHomeScreen() {
```

"Very Secure" email & password credentials!

Possibly via a Web Service?? (Not now obviously)

Update Preferences with data

End Result


Genymotion - Nexus S - 4.3 - API 18 - 480x800 (1...
8:08
CoffeeMat
free for personal use
Genymotion for personal use - S5 - 4.4.2 - API 19 - 1080x192
CoffeeMate 4.0
CoffeeMate
Email
Password
Login
New to CoffeeMate? Register here
You must enter an email & password
free for personal use
Genymotion for personal use - S5 - 4.4.2 - API 19 - 1080x1920 (1080x1...
3:25
CoffeeMate 4.0
CoffeeMate
Email
dd
Password
Login
New to CoffeeMate? Register here
Unable to validate your email & password
free for personal use
Genymotion for personal use – S5 – 4.4.2 – API 19 – 1080...
2:04
CoffeeMate 4.0
Coffee Check In
Search Coffee's
Favourite Coffee's
Recently Added Coffee's
Mocca Latte
Ardkeen Stores
€2.99
4.0 *
Espresso
Tescos Stores
€1.99
3.5 *
Standard Black
Ardkeen Stores
€1.99
2.5 *
Cappuccino
Spar Shop
€1.49
2.5 *
www.jumpyjosh.com
free for personal use

Using Files

File Access (Internal & External)

- Store data to file
- Use java.io.* to read/write file
- Only local file can be visited
  - Advantages: can store large amounts of data
  - Disadvantages: file format changes and/or updates may result in significant programming/refactoring
- Very similar to file handling in java desktop applications
- Generally though, not recommended

Read from a file

- Open a File for input
  - Context.openFileInput(String name)
  - If failure then throw a FileNotFoundException

```
public Map<String,String> readFromFile(Context context){
     Map<String,String> temp = null;

     try{
       inByteStream = context.openFileInput(FILENAME);
            OIStream = new ObjectInputStream(inByteStream);

       temp = (Map<String,String>)OIStream.readObject();

       inByteStream.close();
       OIStream.close();
     }
      catch(Exception e){...}

     return temp;
     }
```

Write to file

- Open a File for output
  - Context.openFileOutput(String name,int mode)
  - If failure then a new File is created
  - Append mode: to add data to file

```
public void writeToFile(Map<String,String> times, Context context){

  try{
      outByteStream = context.openFileOutput(FILENAME, Context.MODE_PRIVATE);
      OOStream = new ObjectOutputStream(outByteStream);
      OOStream.writeObject(times);
      outByteStream.close();
      OOStream.close();
     }
  catch(Exception e){...}

}
```

Write file to SDCard

- To get permission for SDCard r/w in AndroidManifest.xml:

```
<uses-permission    android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"
/>
<uses-permission    android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
```

SDCard read/write

- Need a SD Card, (obviously ☺)

```
if(Environment.getExternalStorageState().equals(Environment.MEDIA_MOUNTED))
{
  File sdCardDir = Environment.getExternalStorageDirectory();
  File saveFile = new File(sdCardDir, "stuff.txt");
  FileOutputStream outStream = new FileOutputStream(saveFile);

  // Same approach as before, once you have a FileOutputStream and/or
  // FileInputStream reference...
  ...

  outStream.close();
}
```

Using ContentProviders

Content Provider

- a content provider is a specialized type of datastore that exposes standardized ways to retrieve and manipulate the stored data.
- Apps can expose their data layer through a Content Provider, identified by a URI.
- Some native apps provide Content Providers
- Your apps can provide Content Providers

Using ContentProvider to share data

- Content Providers are the Android platforms way of sharing information between multiple applications through its ContentResolver interface.
- Each application has access to the SQLite database to maintain their information and this cannot be shared with another application.

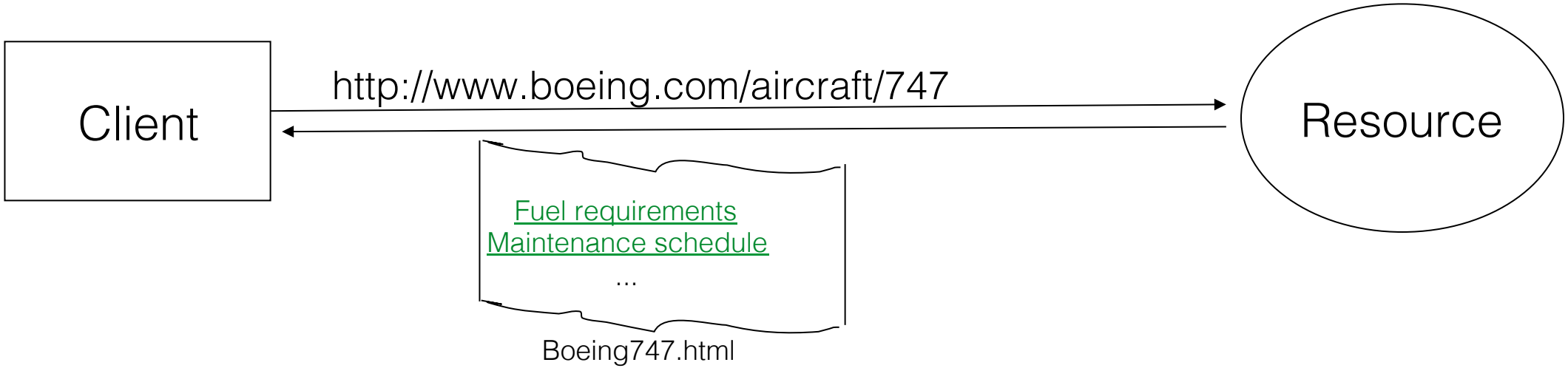
```
public class PersonContentProvider extends ContentProvider{
    public boolean onCreate()
    public Uri insert(Uri uri, ContentValues values)
    public int delete(Uri uri, String selection, String[]
                                selectionArgs)
    public int update(Uri uri, ContentValues values, String
     selection, String[] selectionArgs)
    public Cursor query(Uri uri, String[] projection, String
     selection, String[] selectionArgs, String
sortOrder)
    public String getType(Uri uri)}
```

Addition to the AndroidManifest.xml

- Add the following user permission tag
  <uses-permission android:name="android.permission.READ_CONTACTS" />
- To give your application access to the contacts information.

```
<manifest .... >
    <application android:icon="@drawable/icon"
    android:label="@string/app_name">
     <provider android:name=".PersonContentProvider"
    android:authorities="ie.wit.provider.personprovider"/>
    </application>
</manifest>
```

Why is it called "**Re**presentational **S**tate Transfer"?



- The Client references a Web resource using a URL.
- A **representation** of the resource is returned (in this case as an HTML document).
- The representation (*e.g.*, Boeing747.html) places the client in a new **state**.
- When the client selects a hyperlink in Boeing747.html, it accesses another resource.
- The new representation places the client application into yet another state.
- Thus, the client application **transfers** state with each resource representation.

REST Characteristics

- REST is not a standard (unlike SOAP)
  - You will not see the W3C putting out a REST specification.
  - You will not see IBM or Microsoft or Sun selling a REST developer's toolkit.
- REST is just a **design pattern**
  - You can't bottle up a pattern.
  - You can only understand it and design your Web services to it.
- REST does prescribe the **use** of standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/*etc.* **(Resource Representations)**
  - text/xml, text/html, image/gif, image/jpeg, *etc.* (Resource Types, MIME Types)

REST Principles

- Everything is a resource
- Every resource is identified by a unique identifier
- Use simple and uniform interfaces
- Communication is done by representation
- Be Stateless

- *We'll look at these, and more, next year ☺.*

Using Bundles

The Bundle Class (Saving)

- Override onSaveInstanceState
  - And pass the Bundle to the superclass method

```
protected void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBlah(someData);
}
```

- Called
  - When user rotates screen
  - When user changes language
  - When app is hidden and Android needs the memory
- Not called
  - When user hits Back button
- Note
  - Superclass method automatically stores state of GUI widgets (EditText data, CheckBox state, etc.)

Bundle : Restoring Data

- Override onRestoreInstanceState
  - Pass Bundle to superclass method
  - Look for data by name, check for null, use the data

```
protected void onRestoreInstanceState(Bundle savedInstanceState) {
      super.onRestoreInstanceState(savedInstanceState);
      SomeType data = savedInstanceState.getBlah(key);
      if (data != null) { doSomethingWith(data); }
}
```

- Called
  - Any time app is restarted after onSaveInstanceState
- Note
  - The same Bundle is passed to onCreate.
  - Superclass method automatically restores widget state

The Bundle Class: Details

- Putting data in a Bundle
  - putBoolean, putBooleanArray, putDouble, putDoubleArray, putString, putStringArray, etc.
    - These all take keys and values as arguments. The keys must be Strings. The values must be of the standard types (int, double, etc.) or array of them.
  - putSerializable, putParceleable
    - Lets you store custom objects. Note that ArrayList and most other builtin Java types are already Serializable
- Retrieving data from a Bundle
  - getBoolean, getBooleanArray, getDouble, getDoubleArray, getString, getStringArray, etc.
    - No typecast required on retrieval. Numbers are 0 if no match.
  - **getSerializable, getParceleable**
    - Typecast required on retrieval. Values are null if no match.

Bundle Summary

- Save data in onSaveInstanceState
  - Can put individual pieces of data in the Bundle, or can add a composite data structure.
  - Custom classes must implement Serializable or Parceleable
- Load data in onRestoreInstanceState or in onCreate
  - Look in Bundle for property of given name
  - For Object types, check for null
  - For number types, check for 0 (zero)

Note: Preventing Screen Rotations

- Issue
  - Screen rotations usually require a new layout
  - They also cause the app to be shutdown and restarted
    - Handling this is the topic of this lecture
- Problem
  - What if you do not have landscape layout?
  - Or have not yet handled shutdown and restart?
- Solution
  - Put an entry in AndroidManifest.xml saying that app runs only in portrait mode (or only in landscape mode).

```
<activity android:name=".YourActivity"
          android:label="@string/app_name"
          android:screenOrientation="portrait">
```

More Reading

- JavaDoc: Activity
  - http://developer.android.com/reference/android/app/Activity.html
    - Introductory parts give lots of details
- Chapters
  - Handling Activity Lifecycle Events *and*
  - Handling Rotation
    - From *The Busy Coder's Guide to Android Development* by Mark Murphy.
      - http://commonsware.com/Android/

Sources

- http://en.wikipedia.org/wiki/JSON
- http://www.w3schools.com/json/
- http://www.json.org/
- http://json-schema.org
- http://www.nczonline.net/blog/2008/01/09/is-json-better-than-xml/
- http://en.wikipedia.org/wiki/SOAP_(protocol)
- http://en.wikipedia.org/wiki/REST
- http://stackoverflow.com/questions/16626021/json-rest-soap-wsdl-and-soa-how-do-they-all-link-together